CPM: A Core Model for Product Data

Steven J. Fenves[1], Sebti Foufou[2], Conrad Bock and Ram D. Sriram
Manufacturing Systems Integration Division,
National Institute of Standards and Technology
Gaithersburg, MD 20899-8263

{sfenves, sfoufou, cbock, sriram}@cme.nist.gov

Abstract: *The support of PLM throughout the product life, from the product's conceptualization to its disposal, requires reliable, complete and efficient data models. The Core Product Model (CPM), initially developed at NIST for a number of in-house research projects, has been extended so as to support the full range of PLM information.*

CPM gives equal status to three aspects of a product or artifact: its function, form and behavior. Thus, CPM can support purely functional reasoning about a product in the conceptual stages of design as well as the recording and modeling of its behavior in the post-design stage.

CPM is a generic, abstract model with generic semantics. It is defined as a UML class diagram.. Three levels of CPM models, denoted as the conceptual, intermediate, and implementation models, are described. Extensions of CPM are briefly presented and a short illustrative example is given.

Keywords: *Product data representation, function, form, behavior, PLM, UML class diagrams.*

1 Introduction

The new generation of product modeling concepts and applications is intended to support Product Lifecycle Management (PLM) by providing support for product data creation, use, storage and communication during all stages of product life, from the product's first conceptualization to its disposal.

To succeed in the mission of supporting PLM, applications need reliable, complete and efficient data models. In a PLM environment, actors are submerged in a heterogeneous and voluminous information flow. It is paramount to be able to: (i) filter this information; (ii) structure it; (iii) integrate and control it; and (iv) channel it so that actors receive and manipulate only information pertinent for their task. The correct performance of these four activities can not be ensured without a reliable product data model.

The initial objectives of the work presented was to provide a common data model among four in-house research and development projects at NIST as well as a base-level data

[1] University Professor Emeritus, Carnegie Mellon University and Guest Researcher, NIST
[2] LE2i Laboratory, University of Burgundy, B.P. 47870, Dijon, France, and Guest Researcher, NIST

representation for a multilevel design information flow model [1; 2]. The first version of the Core Product Model (CPM) responding to these objectives was presented in [3].

As interest in PLM intensified, it became increasingly clear that CPM possesses some of the key characteristics needed to support the full range of PLM information. The objective therefore became to expand the CPM to serve as the basic, top-level model for all product realization information [2].

2 The Core Product Model

2.1 Overview

CPM is a generic, abstract model with generic semantics. Semantics meaningful for any particular domain are to be embedded within an implementation model and the policy of use of that model.

CPM is based on two principles. First, the key object in the CPM is the artifact. Artifact represents a distinct entity in a product, whether that entity is a component, part, subassembly or assembly. Second, the artifact is an aggregation of three objects representing the artifact's three principal aspects:

- the artifact's *function* describes what the artifact is supposed to do. The artifact satisfies customer needs and/or engineering requirements largely through its function. The term function is often used synonymously with the term *intended behavior*.
- the artifact's *form* represents the proposed design solution for the design problem specified by the function. The artifact's physical characteristics are modeled in CPM in terms of its *geometry* and *material*, because many of the intended applications tend to treat these two aspects differently.
- the artifact's *behavior* describes how the artifact's form implements its function. Behavior is governed by engineering principles which are incorporated into a behavioral or causal model. Application of the *behavioral model* to the artifact describes or simulates the artifact's *observed behavior*. The observed behavior can then be examined with respect to the requirements to yield the *evaluated behavior.*

The suitability of CPM for supporting the product information needs over the full range of PLM activities is due to the above three-way partition of a product's aspects:

- in the early conceptual stages of design, when the product's form has not yet been selected, the function aspect supports reasoning about the requirements on the product and the allocation of the expected functions of the product; and
- in the post-design stages of manufacturing, operation, and maintenance - when the form of the artifact is unchanged - its behavior (such as manufacturability, operatability, cost or durability) can be modeled, observed, evaluated and recorded in the behavior aspect.

2.2 Components of CPM

CPM consists of two sets of classes, called *object* and *relationship* classes, patterned after the Entity-Relationship model [4], equivalent to the Unified Modeling Language (UML) classes and association classes, respectively [5]. A UML class diagram of CPM is shown in Figure 1. In order to provide a clearer exposition, selected portions of Figure 1 are presented separately in Figures 2 through 5.

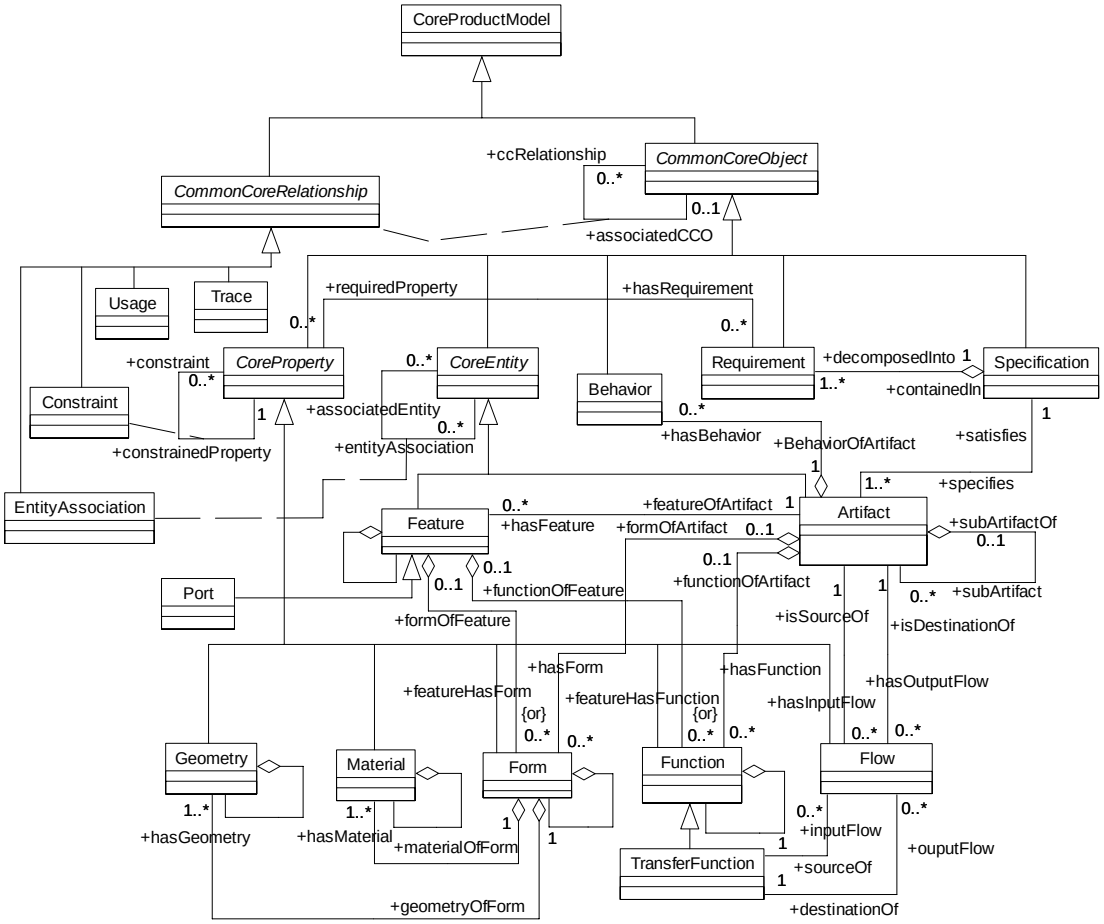

Figure 1. UML class diagram of the Core Product Model

In the text that follows, names of classes are capitalized (e.g., **Information**) and names of attributes are not (e.g., **information**).

2.2.1 Abstract classes

There are five abstract classes (classes with no instances):

- **CoreProductModel** represents the highest level of generalization.

- **CommonCoreRelationship** is the base class for all association classes.
- **CommonCoreObject** is the base class for all object classes.
- **CoreEntity** is the base class from which **Artifact** and **Feature** are specialized.
- **CoreProperty** is the base class from which **Function**, **Flow**, **Form**, **Geometry** and **Material** are specialized.

Figure 2 shows the abstract classes.

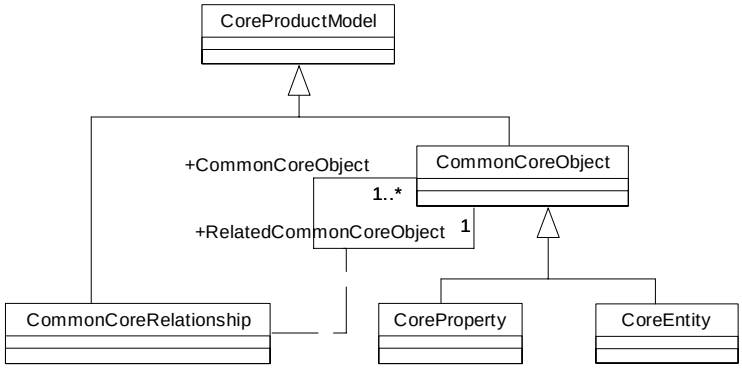

Figure 2. CPM abstract classes

2.2.2 Object classes

There are eleven object classes:

- **Artifact** represents a distinct entity in a product, whether that entity is a component, part, subassembly or assembly.
- **Feature** is a portion of the artifact's form that has some specific function. An artifact may have design features, analysis features, manufacturing features, etc., as determined by their respective functions. **Feature** has its own containment hierarchy, so that compound features can be created out of other features (but not artifacts).
- **Port** is a specialization of **Feature**, sometimes referred to as an interface feature, through which the artifact is connected to (or interfaced with) other artifacts.
- **Function** is what the artifact is supposed to do, that is, its *intended behavior*.
- **TransferFunction** is a specialized form of **Function** involving the transfer or transformation of an input flow into an output flow.
- **Form** of the artifact is the proposed design solution for the design problem specified by the function, represented in terms of its geometry and material.
- **Geometry** is the spatial description of the artifact.
- **Material** is the description of the internal composition of the artifact.
- **Flow** is the medium (fluid, energy, message stream, etc.) that serves as the output of one or more transfer function(s) and the input of one or more other transfer function(s). A flow is also identified by its source and destination artifacts.
- **Behavior** describes how the artifact's form implements its function. Behavior has three specialized attributes:
  - the **behavioralModel**;

  - the **observedBehavior**; and
  - the **evaluatedBehavior**.
- **Specification** represents the collection of information relevant to the design of an artifact deriving from customer needs and/or engineering requirements.
- **Requirement** is a specific element of the **Specification** that governs some aspect of the artifact's function or form. Requirements cannot apply to behavior, which is strictly determined by the behavioral model.

Figure 3 provides a view of the object classes.

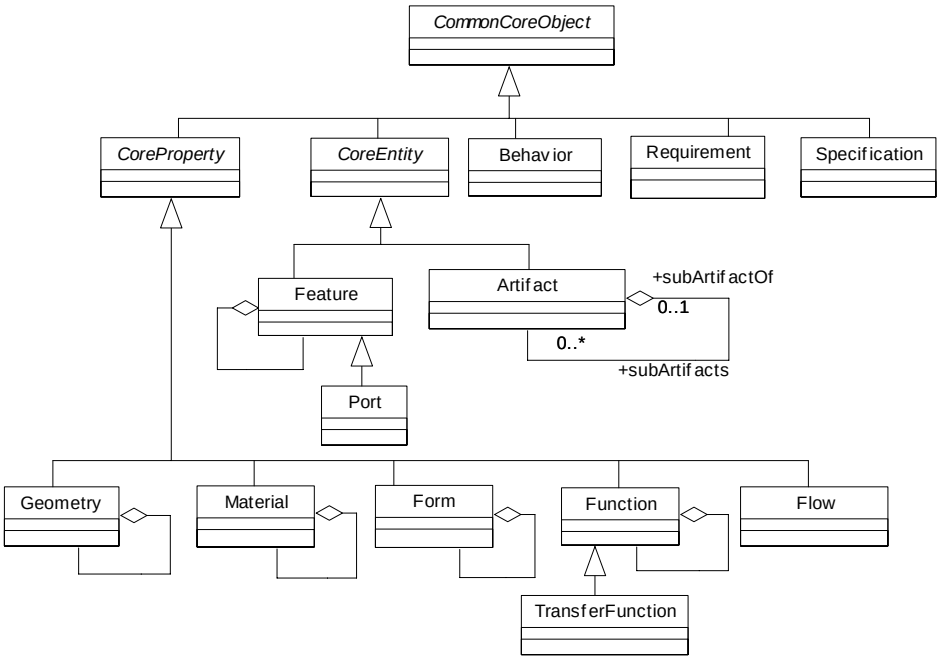

Figure 3. CPM object classes

2.2.3 Relationship classes

There are four relationship classes:

- **Constraint** is a specific shared property of a set of entities that must hold in all cases. In CPM, only the entity instances that constitute the constrained set are identified.
- **EntityAssociation** is a set membership relationship among artifacts, features and ports.
- **Usage** is a mapping from **CommonCoreObject** to **CommonCoreObject,** particularly useful when constraints apply to the specific "target" entity but not to the generic "source" entity, or when the source entity resides in an external catalog or design repository.
- **Trace** is structurally identical to **Usage,** particularly useful when the "target" entity in the current product description depends in some way on a "source" entity in another product description. The **type** attribute of **Trace** specifies the nature of the dependence (*alternative_of, version_of, derived_from, is_based_on*, etc.).

Relationship classes are shown in Figure 4**.**

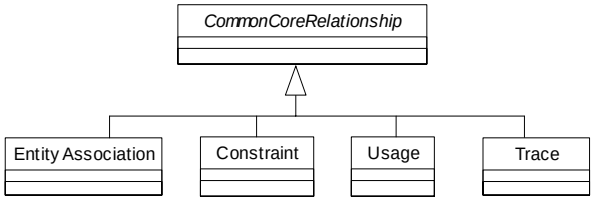

Figure 4. CPM relationship classes

The relationships between object classes are shown in Figure 5.

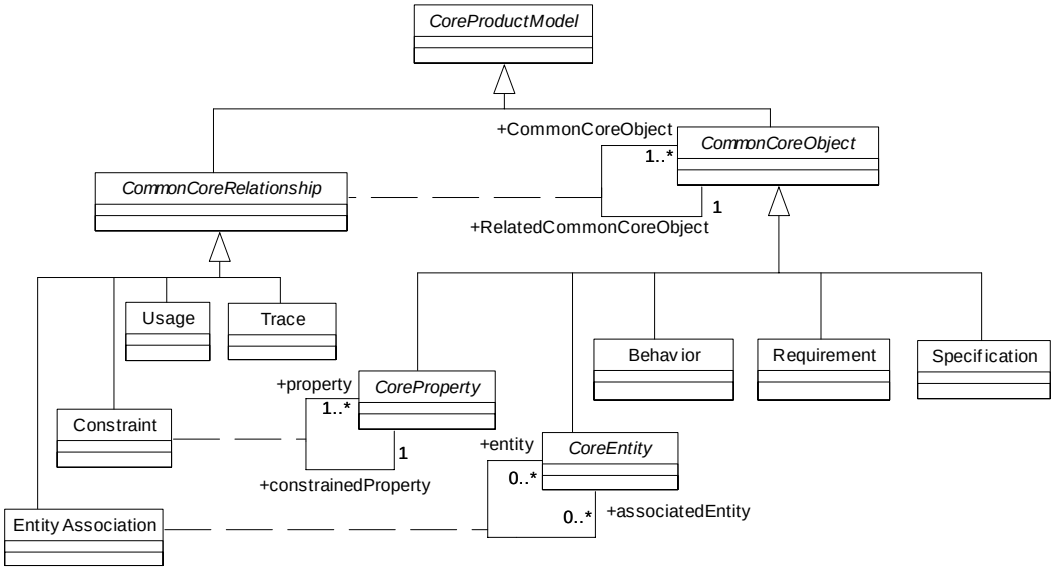

Figure 5. Relationships between object classes

2.2.4 Utility classes

There are three utility classes:

- **Information** is a container consisting of:
  - a textual **description** slot;
  - a textual **documentation** string (e. g., a file path or URL referencing more substantial documentation); and
  - a **properties** slot that contains a set of attribute-value pairs stored as a string.
- **ProcessInformation** is a container for attributes related to the product development process, such as state and level, as used in [1], alternative and/or version designation or other process descriptors.
- **Rationale** records explanatory information on the reasons for or justifications of a particular decision concerning the artifact.

The utility classes are of primary interest in intermediate information models generated from the conceptual CPM, as discussed in Section 4.2.

2.3 Associations and aggregations

The following associations are provided:

- all object classes except **Flow** have their own separate, independent decomposition hierarchies, also known as "partOf" relationships or containment hierarchies[3].
- there are associations between:
  - a **Specification** and the **Artifact** that results from it;
  - a **Flow** and its source and destination **Artifact**s and its input and output **Function**s; and
  - an **Artifact** and its **Feature**s.
- most importantly, four aggregations are fundamental to the CPM:
  - **Function**, **Form** and **Behavior** aggregate into **Artifact**;
  - **Function** and **Form** aggregate into **Feature**;
  - **Geometry** and **Material** aggregate into **Form**; and
  - **Requirement**s aggregate into **Specification.**

3 CPM models

CPM models exist at three levels, denoted as the conceptual, intermediate, and implementation models, described below.

3.1 Conceptual model

CPM is conceived as a conceptual model without domain-specific semantics. Thus, CPM is limited to attributes required to capture generic product information and to create relationships among the classes. CPM intentionally excludes attributes that are domain-specific (e.g., attributes of mechanical or electronic devices) or object-specific (e.g., attributes specific to function, form or behavior).

3.2 Intermediate model

In order to make the conceptual CPM directly usable, two generic information modeling concepts have been adopted so as to be able to create intermediate models.

First, each object and relationship has an **Information** attribute, described in Section 2.2.4. The attribute of interest here is the **properties** slot, where attribute-value pairs record all domain- or object-specific attributes.

[3] For clarity, only the **subArtifact/subArtifactOf** containment hierarchy of **Artifact** is labeled in Figures 1 and 3.

Second, each object and relationship, except for the abstract and utility classes, has an attribute called **type**, the value of which is a string that acts as a symbolic classifier[4]. Each object and relationship class may have a distinct hierarchical taxonomy of terms associated with that class. The value of the **type** attribute corresponds to one of the terms within the taxonomy for the given class. For example, "convert" is one of numerous types of transfer functions and the term can serve as the value of the **type** attribute of an instance of the class.

Using the above two modeling concepts, an artifact instance of type "pin" with specified length and diameter attribute values is represented in the intermediate model as shown in Figure 6.

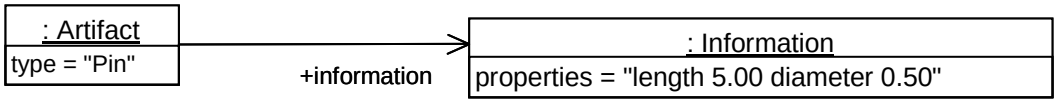

Figure 6. A CPM instance with attributes and values

Such a representation will generally be appropriate and sufficient either for the early conceptual phases of design of an artifact, where typically there is a small number of instances and few attributes of interest for each instance, or for design repositories, where only condensed representations of completed design are stored[5]. Over time, as intermediate models are built up, object and relationship classes in the model may acquire their individual generic engineering classification hierarchies. Eventually, these taxonomies may be expanded into full ontologies of the terms and their semantic relationships. However, this intermediate model will not scale to a full implementation, where thousands of instances may occur, each with a long list of application-specific attributes.

3.3 Implementation models

For application to industrial-scale systems, the conceptual model of CPM must be translated into an implementation model. This is called *model compilation* and is a part of the overall Model-Driven Architecture (MDA) defined by the Object Management Group (OMG) [9]. MDA provides for translation of Platform-Independent Models (PIMs), such as CPM, to Platform-Specific Models (PSMs) and for the generation of efficient implementation languages.

Implementations based on CPM may use the various **type** attributes, their underlying taxonomies and the attribute-value pairs stored in the entities' **properties** slots to provide the means for model compilation of domain-specific specializations of the CPM classes. Specifically, the model compiler would:

[4] The semantics of the term **type** used in this report differs from that of the term "data type" commonly used in computer science data structure definitions. The use of the term **type** in this report is consistent with the definition used in the FRISCO report: "Type (Synonym: 'Category'): A type of things is a specific characterisation (e.g. a predicate) applying to all things of that type" [6]

[5] The NIST Design Repository [7], using a product model closely based on CPM, contains some fairly substantial designs, such as the complete record of the Charters of Freedom enclosures built at NIST [8].

- create subclasses of **Artifact** from the classification hierarchy in the **type** slot; and
- define attributes on the subclasses from the attribute names in the **properties** slot.

These subclasses could be generated into a UML repository [10], as a PIM, then into a compilable language. This provides flexibility in choosing an implementation language.

Finally, model compilation may be used to translate CPM's delegation-style of reusing designs to the type/instance style of computational modeling. CPM uses **Artifact** for the representation of the information at three different stages in the lifecycle of an artifact:

- description of classes of physical objects, for example, the design of a particular kind of gear box;
- use of the above descriptions in composing designs for other physical objects, for example, the use of a particular gear box design in the description of a certain model of car; and
- descriptions of physical objects conforming to the designs above, for example, maintenance record for an individual physical gear box, with serial number 3463, installed in a particular car with VIN number 92345645.

The use of **Artifact** for all three reflects the engineer's viewpoint that they represent different stages in the lifecycle of the same artifact. Each stage may have different attribute values and even different attributes. These stages are differentiated by different values in the instances of the **ProcessInformation** class and related by the **Usage** association in CPM. Computational models, on the other hand, usually have distinct elements for each of the above stages, called *type* (or *class*), *usage* (or *role*), and *instance*.[6] These reflect common information system construction practices of using program development environments to define the shapes of data structures (types, stage 1), and monitoring the execution of those programs in a separate debugging environment to find the actual data stored in those structures (instances, stage 3). Modern modeling techniques introduce usages or roles to more reliably compose designs (usages, stage 2) [12]. Model compilers can bridge the engineering and computational viewpoints by storing the rules by which the three stages are distinguished in the engineering model, using these to categorize artifacts, and generate the corresponding computational models.

4 Extensions of the CPM

The following extensions to the CPM have been reported:

- the **Open Assembly Model** (OAM) provides a standard representation and exchange protocol for assemblies [13-15]. The assembly model defines both a system level conceptual model and the associated hierarchical relationships. The model provides a way for tolerance representation and propagation, kinematics representation, and engineering analysis at the system level.

[6] Some computational models use one element as CPM does, but distinguish the three stages by a special attribute, for example, MOOD in the Health Level 7 Reference Information Model [11].

- the **Product Semantic Representation Language** (PSRL) utilizes CPM for the development of a formal representation of product information [16]. Formal description logic (OWL) is used to encode the PSRL.
- the **Design-Analysis Integration** project proposes a conceptual data architecture that can provide tighter integration of spatial and functional design and support analysis-driven design and opportunistic analysis [17]. CPM serves as the organizing principle of the Master Model from which discipline-specific functional models (views) are idealized.
- the **Product Family Evolution Model** extends CPM to the representation of the evolution of product families and of the rationale of the changes involved [18]. The model represents the independent evolution of products and components through families, series and versions, and the rationale for the changes.
- the **Heterogeneous Material Model** extends CPM to components with continuously varying material properties [19]. Distance fields are associated with a set of material features, where values and rates of material properties are specified.
- the **Mechatronic Device Model** is a framework supporting the conceptual design of multiple interaction-state mechatronic devices, where the interactions between the use-environment and the device can have different qualitative structures [20]. Devices within a state are modeled by extensions of CPM.
- the **Embedded System Model** is feature-based approach to the co-design of hardware and software in embedded systems [21]. The approach defines extensions to CPM providing a representation for the embedded system feature model.

Extensions and implementations of CPM may explicitly assign attributes to specializations of the CPM objects and relationships so as to provide interoperability with new systems, legacy data models such as STEP, or existing CAD programs.

5. Illustrative example

The planetary gear system example considered in this section was presented in detail in [14], where it was used to illustrate the representation of both the **Artifact** containment hierarchy and the assembly associations comprising the Open Assembly Model. Our interest here is to show how CPM captures design information about a product; thus, only data important from the design point of view are modeled.

Figure 7 shows the components of the planetary gear system (PGS): the main artifact is the planetary gear; it is composed of 13 subartifacts: the output housing, the input housing, the ring gear, the sun gear, the planet gear carrier and eight screws. Information pertaining to the function, form, behavior and specification related to these subartifacts is not presented here.

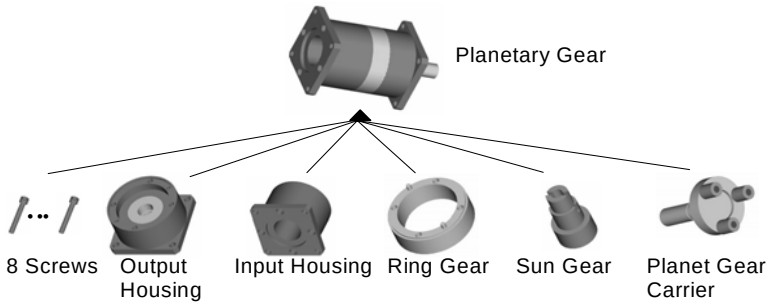

Figure 7. The planetary gear system

Figure 8 shows an instance diagram for the representation of the gear system in CPM. The figure shows only the structure of the PGS as an instance of the **Artifact** class. This instance is linked through subArtifact relationships to a set of other instances representing the subartifacts of the PGS. The figure also shows instances of the **Function**, **Form**, **Behavior, Specification**, and **Feature** classes. The contents and values of the attributes of these instances are not shown.

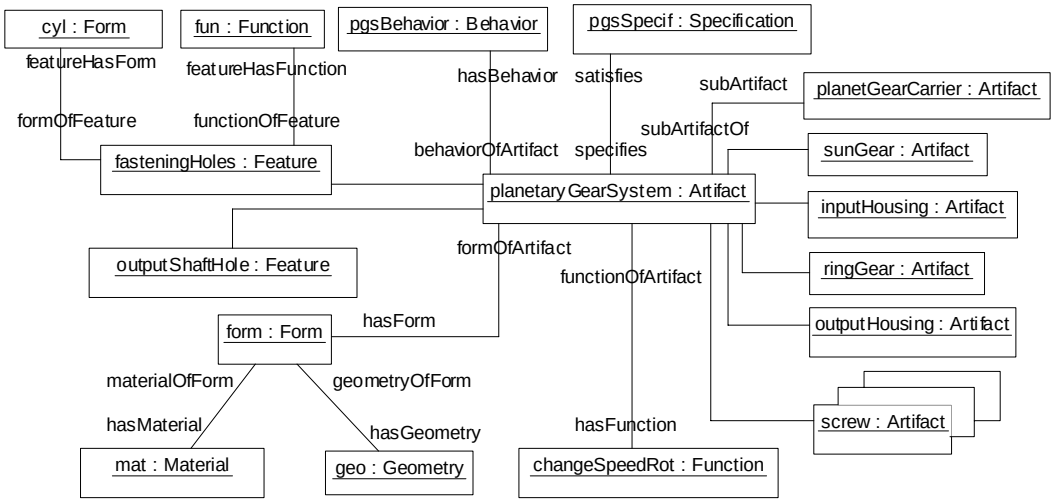

Figure 8. CPM instance diagram for the planetary gear system

6. Related work

CPM follows the tradition of work in the area of artifact representation. The division of artifact information into the categories of form, function, and behavior has its roots in earlier work in intelligent design systems. The model is most directly descended from the representation developed as part of the NIST Design Repository project [7].

The MOKA project shares the motivation of the CPM [22]. MOKA exploits STEP, KIF and other DARPA-related efforts to create an integrated product model for knowledge

based engineering. The MOKA modeling language (MML), based on UML, is designed to represent engineering design knowledge at a user level for deployment in KBE applications. MOKA goes one step further than CPM towards the implementation models discussed in Section 4.3 by using the MOCA metamodel as the prototype of domain-specific application models. The MOKA Product Model supports five distinct views of a product:

- **Structure** defines the hierarchical decomposition of a product's structure into parts, assemblies, and features. Structure can be physical, logical or conceptual at any stage of design;
- **Function** defines the functional decomposition of the product and principles of solution;
- **Behavior** includes a state model of the various states of a product and of the transition from one state to another;
- **Technology** includes materials and manufacturing process information; and
- **Representation** includes any other user-defined technological information, including alternate representations of the physical structure.

The similarly motivated Virtual Product Model is a comprehensive effort at developing standards for different aspects of product modeling, including functional decomposition, product data, design process information and other aspects of product development [23].

The IMPROVE project is a similar large scale effort in the area of process systems engineering, where design process information models are integrated with process system design information models [24].

7. Summary and conclusions

CPM is a generic, abstract model with generic semantics. It gives equal status to three aspects of a product or artifact: its function, form and behavior. Thus, CPM can support: (i) purely functional reasoning about a product in the conceptual stages of design; (ii) recording and analysis of its behavior in the post-design stages; as well as (iii) the "traditional" engineering design activities of generating the product's form in response to the specified function, evaluating the product's behavior by analysis and simulation, and modifying the form until the behavior satisfies the function.

At present, CPM exists as a conceptual model. Extensions at the same conceptual level have illustrated that CPM is readily expandable, although some conflicts have been detected (for example, in some domains features do have independent behavior, not supported by CPM; in other domains, geometry and material are not the appropriate top-level specializations of form). A few intermediate level models have been generated, primarily for illustrative purposes.

The extent to which CPM can serve as the central information support mechanism for PLM will be determined by the implementation models developed from the conceptual CPM. As indicated in Section 3.3, an extension or application of Model-Driven Architecture will be needed to provide both: (i) translation of platform-independent

models to platform-specific models; and (ii) expansion of classes and their attributes from the domain-independent CPM model to models for specific domains.

References

1. Shooter, S. B., Keirouz, W. T., Szykman, S., and Fenves, S. J., "A Model for the Flow of Design Information in Product Development," *Engineering with Computers*, Vol. 16, 2000, pp. 178-194.

2. Szykman, S., Fenves, S. J., Keirouz, W. T., and Shooter, S., "A foundation for interoperability in next-generation product development systems," *Computer-Aided Design*, Vol. 33, No. 7, 2001, pp. 545-559.

3. Fenves, S. J., "A Core Product Model For Representing Design Information," National Institute of Standards and Technology, NISTIR 6736, Gaithersburg, MD 20899, USA, 2001.

4. Chen, P. P., "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems*, Vol. 1, No. 1, 1976, pp. 9-36.

5. Booch, G., Rumbaugh, J., and Jacobson, I., *The United Modeling Language User Guide*, Addison-Wesley 1997.

6. Verrijn-Stuart A A. FRISCO - A framework of information system concepts - The Revised FRISCO Report (Draft January 2001), IFIP WG 8.1 Task group FRISCO. 2001.

7. Szykman, S., Racz, J. W., Bochenek, C., and Sriram, R. D., "A Web-based System for Design Artifact Modeling," *Journal of Design Studies*, Vol. 21, No. 2, 2000, pp. 145-165.

8. Allen, R. H., Sriram, R. D., Szykman, S., and Fijol, R. J., "Representing the Charters of Freedom Encasements in a Decision Repository: A Case Study," Pittsburgh, Pennsylvania, 2001.

9. OMG. Model-Driven Architecture. 2004.

10. Bock, C., "UML without Pictures," *IEEE Software Special issue on Model-driven Development*, 2004.

11. Health Level 7. HL7 Reference Model. 2004.

12. OMG. UML 2.0 Superstructure Specification. 2003.

13. Sudarsan, R., Baysal, M. M., Roy, U., Foufou, S., Bock, C., Fenves, S. J., Eswaran, E., Lyons, K. W., and Sriram, R. D., "Information Models for Product Representation: Core and Assembly Models," NISTIR 7173, NIST, Gaithersburg, MD 20899, Dec. 2004.

14. Sudarsan, R., Young-Hyun, H., Feng, S. C., Roy, U., Fujun W., Sriram, R. D., and Lyons, K. W., "Object-oriented Representation of Electro-Mechanical Assemblies Using UML," National Institute of Standards and Technology, NISTIR 7057, Gaithersburg, MD 20899, USA, 2003.

15. Sudarsan, R., Young-Hyun, H., Foufou, S., Feng, S. C., Roy, U., Fujun W., Sriram, R. D., and Lyons, K. W., "A Model for Capturing Product Assembly Information ," *to appear in Journal of Computing and Information Science in Engineering*, 2005.

16. Patil, L., Dutta, D., and Sriram, R. D., "Ontology-based Exchange of Product Data Semantics," *IEEE Transactions on Automation Science and Engineering*, Vol. 2, No. 3, 2005, pp. 213-255.

17. Fenves, S. J., Choi, Y., Gurumoorthy, B., Mocko, G., and Sriram, R. D., "Master Product Model for the Support of Tighter Design-Analysis Integration," National Institute of Standards and Technology, Gaithersburg, MD 20899, NISTIR 7004, 2003.

18. Wang, F., Fenves, S. J., Sudarsan, R., and Sriram, R. D., "Towards Modeling the Evolution of Product Families," Proceedings of the 2003 ASME Design Engineering Technical Conferences, Chicago, IL, 2003.

19. Biswas, A., Fenves, S. J., Shapiro, V., and Sriram, R. D., "Representation of Heterogeneous Material Properties in the Core Product Model, 2005.," *Engineering with Computers*, 2005.

20. Xu, C., Gupta, S. K., Yao, Z., Gruninger, M., and Sriram, R. D., "Toward Computer-Aided Conceptual Design of Mechatronic Devices with Multiple Interaction-States," 2005.

21. Zha, X. F., Fenves, S. J., and Sriram, R. D., "A Feature-based Approach to Embedded System Hardware and Software Co-Design," 2005.

22. MOKA. MOKA: A Framework for structuring and representing engineering knowledge. http://www.kbe.coventry.ac.uk/moka/miginfo.htm . 1999.

23. Ingward, B., Falk, M., Edwin, S., and Erik, M.. Project iViP: Integrated Virtual Product Creation - Final Report. Frank-Lothar, K, Trac, T, and Irich, A. 2002. Organisation responsible for production and manufacturing technologies (PFT), Research Centre Karlsruhe GmbH.

24. Marquardt, W. and Nagl, M., "Workflow and Information Centered Support of Design Processes," *Computers & Chemical Engineering*, Vol. 29, No. 1, 2004, pp. 65-82.